METHOD FOR MANIPULATING DATA IN A GROUP OF PROCESSING ELEMENTS TO TRANSPOSE THE DATA USING A MEMORY STACK

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] The present application is related to the following applications: Method for Manipulating Data in a Group of Processing Elements to Perform a Reflection of the Data (docket no. DB001071-000); Method for Manipulating Data in a Group of Processing Elements to Transpose the Data (docket no. DB001070-000); Method for Manipulating the Data in a Group of Processing Elements (docket no. DB001072-000); and Method of Rotating Data in a Plurality of Processing Elements (docket no. DB001063-000), all filed concurrently herewith.

BACKGROUND OF INVENTION

[0002] The present invention relates generally to parallel processing and, more specifically, to parallel processing in an active memory device or single instruction, multiple data (SIMD) computer.

[0003] A single, synchronous dynamic random access memory (SDRAM) chip has an internal data bandwidth of greater than 200 G bits/s and a very wide data bus (thousands of bits). That vast data bandwidth provides an opportunity for high performance. Active memories represent one effort to use that vast data bandwidth to improve performance.

[0004] An active memory is a memory device which has a built in processing resource. One of the principal advantages of active memory is that data is processed close to where it is stored. Usually the processing resource is a highly parallel computer system which has processing power to exploit the very high data bandwidths available inside a memory system. An example of an active memory system is illustrated in Figure 1.

[0005] In FIG. 1, a main memory 10 appears as a traditional memory to a CPU 12 except that the main memory 10, by virtue of memory processors 14, can be instructed to perform tasks on its data without the data being transferred to the CPU 12 or to any other part of the system over a system bus 16. The memory processors 14 are a processing resource distributed throughout the main memory 10. The processing resource is most often partitioned into many similar processing elements (PEs). The PEs are usually simple and operate in parallel. In such a system, the work of the CPU 12 is reduced to various operating system tasks such as scheduling. A substantial portion of the data processing is performed within the main memory 10 by virtue of the memory processors 14.

[0006] Active memory systems have a long history. The earliest systems were built in the 1960's. However, until the advent of integrated logic and current DRAM technologies, active memory computers were always expensive, special machines, excluded from mass market applications. For active memory to be effective, the organization of data in the PE array is an important consideration. Hence, the provision of an efficient mechanism for moving data from one PE to another is an important consideration in the design of the PE array.

[0007] In the past, several different methods of connecting PEs have been used in a variety of geometric arrangements including hypercubes, butterfly networks, one-dimensional strings/rings and two-dimensional meshes. In a two-dimensional mesh or arrays, the PEs are arranged in rows and columns, with each PE being connected to its four neighboring PEs in the rows above and below and columns to either side which are sometimes referred to as north, south, east and west connections.

[0008] Disclosed in G.B. Patent Application Serial No. GB02215 630, entitled Control of Processing Elements in Parallel Processors, filed September 17, 2002 is an arrangement in which a column select line and a row select line can be used to identify processing elements which are active, e.g. capable of transmitting or receiving data. The ability to use a row select signal and a column select signal to identify active PEs provides a substantial advantage over the art in that it enables data to be moved through the array of PEs in a nonuniform manor. However, the need still exists for enabling PEs within the array to work independently of its neighboring PEs even though each PE within the array has received the same instruction.

SUMMARY OF THE INVENTION

[0009] The present invention is directed to a method for transposing data in a plurality of processing elements comprising a plurality of shifting operations and a plurality of storing operations. The shifting and storing operations are coordinated to enable data to be stored along a diagonal of processing elements from a first pair of directions and to be output from the diagonal in a second pair of directions perpendicular to the first pair of directions. The plurality of storing operations are responsive to the processing elements' positions. The first and second pairs of directions are selected from among the dimensions of the array, e.g., the +x/-x, +z/-z and +y/-y pairs of directions.

[0010] The present invention is also directed to a method for transposing data in a plurality of processing elements comprising a first shifting of data in a first direction followed by a first storing of data along a diagonal of processing elements in response to the first shifting. A second shifting of data from the diagonal is performed so as to output the stored data from the diagonal; the second shifting being in a second direction perpendicular to the first direction. A second storing of data is performed by at least certain of the processing elements in response to the second shifting and in response to the processing element's position. A third shifting of data in a third direction opposite to the first direction is followed by a third storing of data along the diagonal in response to the third shifting. A fourth shifting of data from the diagonal is performed so as to output the data stored in the diagonal in response to the third shifting; the fourth shifting of data being in a fourth direction opposite to the second direction. Finally, a fourth storing of data by at least certain other of the processing elements is performed in response to the fourth shifting and in response to the processing element's position.

[0011] The present invention contemplates hardware, e.g., memory containing an ordered set of instructions, for carrying out the disclosed methods. The present invention provides an efficient method for obtaining the transpose of the data in a plurality of interconnected processing elements. Those advantages and benefits, and others, will be apparent from the description of the invention appearing below.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] For the present invention to be easily understood and readily practiced, the present invention will be described in conjunction with an exemplary embodiment, for purposes of illustration and not limitation, in conjunction with the following figures wherein:

[0013] FIG. 1 is a block diagram illustrating the concept of active memory;

[0014] FIG. 2 is a high level block diagram of one example of an active memory on which the methods of the present invention may be practiced;

[0015] FIG. 3 is a high level block diagram of one example of a PE;

[0016] FIG. 4 is a diagram illustrating one type of logic circuit that may be used to interconnect the PE illustrated in FIG. 3 to other PEs;

[0017] FIG. 5 illustrates one method of interconnecting PEs to form an array of PEs;

[0018] FIGs. 6A and 6B illustrate one example of an edge shift;

[0019] FIGs. 7A and 7B illustrate one example of a planar shift;

[0020] FIGs. 8A and 8B illustrate one example of a wrap shift;

[0021] FIGs. 9A and 9B illustrate one example of a vector shift;

[0022] FIGs. 10A and 10B illustrate another example of a vector shift;

[0023] FIGs. 11A and 11B illustrate one example of a data broadcast from the edge registers in which a row and column select function enabled;

[0024] FIGs. 12A and 12B illustrate one example of a broadcatch in which only one column is selected;

[0025] FIGs. 13A and 13B illustrate one example of selected edge registers being loaded with the AND of selected columns;

[0026] FIGs. 14A and 14B illustrate another example of a data broadcast;

[0027] FIG. 15 illustrates an initial matrix of data;

[0028] FIGs. 16A and 16B illustrate two possible transpositions of the data shown in FIG. 15;

[0029] FIGs. 17A through 17D illustrate the steps of one embodiment of the method of the present invention for obtaining the transposed data of FIG. 16A;

[0030] FIGs. 18A through 18D illustrate the steps of another embodiment of the method of the present invention for obtaining the transposed data of FIG. 16B;

[0031] FIGs. 19A and 19B illustrate the steps of another embodiment of the method of the present invention for obtaining the transposed data of FIG. 16A, and

[0032] FIGs. 20A through 20B illustrate the steps of another embodiment of the method of the present invention for obtaining the transposed data of FIG. 16A.

DESCRIPTION OF THE INVENTION

[0033] Illustrated in FIG. 2 is a high level block diagram of one example of an active memory device 18 on which the methods of the present invention may be practiced. The reader should understand that the methods of the present invention are generally applicable to any group of processing elements having the necessary physical connections between PEs to enable the manipulation of data as required by the methods. The hardware illustrated in FIG. 2 is disclosed for purposes of illustration and not limitation. Furthermore, those of ordinary skill in the art will recognize that the block diagram of FIG. 2 is an overview of an active memory device with a number of components known in the art being omitted for purposes of clarity.

[0034] The active memory device 18 of FIG. 2 is intended to be deployed in a computer system as a slave device, where a host processor (e.g. CPU 12 in FIG. 1) sends commands to the active memory device 18 to initiate processing within the active memory device 18. A complete processing operation, i.e., data movement and processing, in the active memory device 18 will usually consist of a sequence of many commands from the host to the active memory device 18.

[0035] The active memory device 18 may have two interfaces, a bus interface 20 and a host memory interface 22, for interfacing with the host or other external logic for the purposes of data input, data output and for control. The host memory interface 22 (data input/output ports) of the active memory device 18 is similar in its operation to the interface of a synchronous DRAM. To access data within a DRAM array 24, the host must first activate a page of data. Each page may contain 1024 bytes of data and there may be 16384 pages in all. Once a page has been activated, it can be written and read through the, for example, 32 bit data input/output ports. The data in the DRAM array 24 is updated when the page is deactivated.

[0036] In the active memory device 18 the input and output ports are separate, or they may be combined into a single bi-directional input/output port. A control output may be provided to control a bi-directional buffer servicing the single bi-directional input/output port.

[0037] The host memory interface 22 may operate at twice the frequency of the master input clock. A copy of the 2x clock may be driven off-chip as a timing reference. Unlike a traditional DRAM, the access time for the host memory interface 22 port takes a variable number of cycles to complete an internal operation, such as an activate or deactivate. A ready signal (rdy) is provided to allow the host to detect when the command has been completed.

[0038] The control or command port (cmd) may be a straightforward 32 bit synchronous write/read interface. Writes place both data and the corresponding address into a FIFO 26 of a task dispatch unit 28, which holds the commands until they are executed in the order they were issued. This arrangement allows a burst of commands to be written to the active memory device 18 suiting the burst operation of many peripheral buses. Reads may operate directly.

[0039] The command port is also synchronous, running at the same frequency as the master input clock. Similarly to the host memory interface 22 port, the clock may be driven out as a timing reference.

[0040] In addition to the two address-mapped ports, the active memory device 18 has an interrupt output (intr) which is used to alert its host to various different conditions.

[0041] Internal control in the active memory device 18 is handled by three processors. The task dispatch unit 28 (mentioned above) receives commands from the command port, interprets them and passes them on to the other two processors, a DRAM control unit 30 and an array sequence controller 32. The task dispatch unit 28 also maintains the addresses for operand placement in an array processor register file (RF) and enables access to on-chip resources such as a program memory 34.

[0042] The DRAM control unit 30 controls the DRAM array 24. The DRAM control unit 30 arbitrates between requests for DRAM array 24 access from the host through host memory access registers (H) and through the host memory interface 22. The DRAM control unit 30 also schedules DRAM array 24 refreshes.

[0043] The array sequence controller 32 controls an array or two dimensional mesh of PEs 36. The sequence controller 32 also executes a program from the program memory 34 and broadcasts control signals into the array of PEs 36. The DRAM control unit 30 and array sequence controller 32 may have a synchronization mechanism, whereby they can link the execution of tasks in either processor.

[0044] The active memory device 18 may contain, according to one embodiment, sixteen 64k x128 eDRAM cores. Each eDRAM core is closely connected to an array of sixteen PEs, making 256 (16 x 16) PEs in all.

[0045] FIG. 3 is a high level block diagram of one example of a PE 37. The PE 37 is comprised of a set of Q registers and a shift network 38 which interact with a set of M registers and another shift register 40. One of the sets of registers and shift network 38, 40 receives inputs from various registers, such as register R0, R1, R2 and 0. The output of the registers and shift networks 38, 40 is input to an arithmetic logic unit (ALU) 42. The ALU 42 is capable of performing various arithmetic functions on its input such as addition, subtraction, etc. as is known. The ALU 42 is in communication with condition logic 44 and a result pipe 46.

[0046] The result pipe 46 is a series of interconnected registers R0, R1, R2 and a neighborhood connection register X, which may be used to output a final value. The result pipe 46 also receives through a multiplexer 47 data in the form of an output signal X from its four neighbors, one to the north (XN), one to the east (XE), one to the south (XS) and one to the west (XW). If the PE 37 happens to be located on an edge of an array, then it may be receiving data from an edge register or a PE in the same row or column, but on an opposite edge, as will be described in greater detail below.

[0047] The result pipe 46 is in communication with a register file (RF) 48 which in turn is in communication with an interface 50. The interface 50 may include a DRAM interface 52 as well as access to the host memory access registers (H).

[0048] The reader should recognize that the PE 37 illustrated in FIG. 3 is exemplary only and is not intended to limit the present invention. For example, the number and location of registers and shift networks may vary, the complexity of the ALU 42 and condition logic 44 may vary, the number of registers and interconnection of registers in the result pipe 46, the size and number of register files, and connection to neighboring PEs as well as other logic may be varied while remaining within the scope of the present invention. The particular architecture illustrated in FIG. 3 was selected to provide a rich register set to enable fairly complex multi-byte operations to be kept within the PE as much as possible.

[0049] For example, the Q registers and shift network 38 allow for data shifting within the 32 bits of the Q register to the left (most significant direction) one, two, four or eight places and eight places to the right as well as for merging data back into a floating point format. The M registers and shift network 40 allow for data shifting within the 32 bits of the M register to the right (least significant direction) one, two, four or eight places and for demerging data from floating point into a signed magnitude plus exponent format. The result from the ALU 42 can be loaded into register R0 while the input from the register file 48 can be loaded into either register R1 or register R2. The neighborhood connection register X can be used as a flexible member of the result pipeline allowing a pipeline length of up to four to be programmed within the PE 37. The X register can be loaded from the R0, R1 or R2 registers, or from the neighborhood interconnection input (the X register of a neighboring PE). The output of the X register can be fed back into the result pipeline at R1 or R2. The register file 48 may be implemented as a 128 entry by 8 bit register file implemented as a synchronous static RAM.

[0050] The DRAM interface 52 may contain two registers, a RAM IN register and a RAM OUT register. Input from the DRAM 24 of FIG. 2 may be held in the RAM IN register while output to the DRAM 24 is held in the RAM OUT register. The RAM IN and RAM OUT registers may reside in the clock domain of the DRAM 24 which typically uses a slower or divided clock derived from the same source as the clock used for the PE array 36. The RAM IN and RAM OUT registers may be controlled directly from the DRAM control unit 30 and are not visible to the programmer. Data can be transferred into and out of the register file 48 using stolen cycles. Data can also be transferred to/from the host memory access registers (H) without stealing cycles from processing in the PE 37.

[0051] Eight host memory access registers (H) may be provided which allows for a short burst of four or eight bytes to be transferred into or out of the DRAM 24 for host access. Those registers may be multiplexed and be visible from the host memory interface 22 (see FIG. 1) as a page of data. More details about the PEs may be found in G.B. Patent Application No. ___________ entitled Host Memory Interface for a Parallel Processor and filed September 17, 2002, which is hereby incorporated by reference.

[0052] FIG. 4 is a diagram illustrating one type of logic circuit that may be used to interconnect PEs of the type illustrated in FIG. 3. The reader will understand that many types of logic circuits may be used to interconnect PEs depending upon the functions to be performed. Using the logic circuit of FIG. 4 to interconnect PEs may result in an array of PEs 36 of the type illustrated in FIG. 5.

[0053] Turning now to FIG. 5, the X register within the result pipe 46 of each PE is driven out as, for example, an eight bit wide X output. Eight bits has been chosen in connection with this architecture as the data width for the PE-PE interconnect to keep a balance between the data movement performance of the array and the improved computational performance. Other sizes of interconnects may be used. The X output is connected to the neighboring inputs of each PE's closest neighbors in the north and west directions. To the south and east, the X output is combined with the input from the opposite direction and driven out to the neighboring PE.

[0054] At the edges of the array 36, the out-of-array connection is selected though a multiplexer to be either the output from the opposite side of the array or an edge/row register 54 or an edge/col. register 56. The edge registers 54, 56 can be loaded from the array output or from the controller data bus. A data shift in the array can be performed by loading the X register from one of the four neighboring directions. The contents of the X register can be conditionally loaded on the AND gate of the row select and column select signals which intersect at each PE. When the contents of the X register is conditionally loaded, the edge registers 54, 56 are also loaded conditionally depending on the value of the select line which runs in the same direction. Hence, an edge/row register 54 is loaded if the column select for that column is set to 1 and an edge/col register 56 is set if the row select is set to 1. The reader desiring more information about the hardware configuration illustrated in FIG. 5 is directed to G.B. Patent Application GB02215 630, entitled Control of Processing Elements in Parallel Processors filed September 17, 2002, which is hereby incorporated by reference.

[0055] With the hardware previously described, a number of shifting operations may be performed as illustrated in FIGs. 6A, 6B through 10A, 10B. In FIG. 6A and 6B, an edge shift is illustrated. In the edge shift, the edge/col registers 56 are active as the data is shifted left to right (west to east) as shown in FIGs. 6A, 6B. The reader will recognize that an edge shift may be performed in the other direction, right to left (east to west). Alternatively, edge shifts may be performed by using the edge/row registers 54 in a north to south or south to north direction.

[0056] Illustrated in FIGs. 7A, 7B is a planer shift. In the planer shift there is no wrap around from the edge of the array. The reader will recognize that in addition to the planer shift illustrated in FIGs. 7A, 7B, planer shifts from east to west, north to south, and south to north may also be performed.

[0057] Illustrated in FIGs. 8A, 8B is a wrap shift. In the wrap shift, the edge/col registers 56 do not participate. Additionally, wrap shifts from east to west, north to south and south to north may be performed.

[0058] Illustrated in FIGs. 9A, 9B is a vector shift. Again, the edge/col registers 56 do not participate. Furthermore, the output of the PE in the bottom right corner of the array wraps to the input of the PE in the upper left corner of the array. In FIGs. 10A and 10B, a vector shift in the direction opposite to the direction of FIGs. 9A, 9B is illustrated. The reader will recognize that vector shifts from north to south and south to north may also be performed.

[0059] Returning to FIG. 5, the PE-PE interconnect may also provide a broadcast and broadcatch network. Connections or buses 58 extend north to south from a column select register 59 and connections or buses 60 extend west to east from a row select register 61. Also provided is row broadcast/broadcatch AND chain 62 and a column broadcast/broadcatch AND chain When used for data broadcast or broadcatch, these connections (column buses 58 and row buses 60) act as if driven by open drain drivers; the value on any bit is the wire-AND of all the drivers outputs. Three control signals (broadcatch, broadcast and intercast) determine the direction of the buses as follows:

- If broadcatch is set to 1, any PE for which the corresponding bits of the row select register 61 and column select register 59 are both set will drive both the row buses 60 and the column buses 58. Note that if no PEs in a row or column drive the bus, the edge register at the end of that row or column will be loaded with 0 x FF.
- If broadcast is set to 1, the row bus 60 is driven from the row select register 61 and the column bus 58 is driven from the column select register 59 and any PE for which the corresponding bits of the row select register 61 and column select register 59 are both set will be loaded from one of the row or column inputs, according to which is selected.
- If intercast is set to 1, any PE in which its A register is 1 will drive its output onto its row bus 60 and column bus 58 and any PE for which the corresponding bits of the row select register 61 and column select register 59 are both set will be loaded from one of the row buses 60 or column buses 58, according to which is selected.

[0060] With the aforementioned connections, a number of operations are possible, some of which are illustrated in FIGs. 11A, 11B through 14A, 14B.

[0061] In FIGs. 11A, 11B, data is broadcast from edge/col registers 56 with the column select register 59 and row select register 61 set as illustrated in FIG. 11A. As a result, data is latched in the PEs as shown in FIG. 11B in which four PEs are active, and the remainder of the PEs in the array are inactive.

[0062] In FIGs. 12A, 12B, a broadcatch instruction is illustrated in which one column is selected by setting the value for that column's bus 58 to 1. In this broadcatch-column operation, only those edge /col. registers 56 for which the row select register 61 bits are set, will be loaded. Similarly, in a broadcatch-row operation (not shown), only those row/edge registers 54 for which the corresponding column select register 59 bits are set, will be loaded.

[0063] FIGs. 13A, 13B illustrate a broad catch instruction. In the illustrated example, the column select register 59 and row select register 61 are used to select the PEs whose values will be AND'ed together and loaded in the corresponding edge/col registers 56. In FIGs. 13A, 13B, the column edge registers 56 are loaded with the AND of selected columns, except where the row select is 0.

[0064] In FIGs. 14A, 14B, an example of an intercast operation is illustrated. In an intercast operation, PEs which drive onto the row buses 60 and column buses 58 are determined by each PE's A register value. The PEs which are loaded are determined by the row and column selects, just like for a broadcast. In FIG. 14A, data is broadcast from the X registers of those PEs where A equals 1 while in FIG. 14B, the column select register 59 and row select register 61 together activate those PEs into which data will be written.

[0065] Using the aforementioned instructions or operations, a group of instructions may be combined into an instruction set for manipulating data within the array 36 of PEs. The instruction set may include a single instruction or operation or a combination of instructions. Each individual instruction is carried out though a series of shifts.

[0066] In operation, an input matrix of data is placed on the shift network, and moved around by using a combination of north, south, east and west shifts. In addition, the column select register 59 and row select register 61 may be used to determine which of the PEs is active. The exact combination of active PEs, instructions, and direction in which the instruction (shift) is performed will depend upon the particular array manipulation required. As the instructions are executed and the shifting proceeds, each PE will be presented with different array values. For example, if a wrap shift is performed a number of times equal to the number of PEs in a row, each PE in the row will see every value held by all of the other PEs in the row.

[0067] A PE can conditionally select any of the values it sees as its final output value by conditionally loading that value, which is representative of an output result matrix. However, only one value, the desired result, is loaded.

[0068] All X values are passed through the PE; the required output value is conditionally loaded once it has arrived in the PE. The conditional loading can be done in various ways. e.g. by using any PE registers except X, R1, or R2. An example is shown below.

Clock	PE C + 0		PE C + 1		PE C + 2		PE C + 3
Cycle							
T + 0	X <= xe(east)	⇐	X <= xe	⇐	X <= xe	⇐	X <= xe
	⇓		⇓		⇓		⇓
T + 1	R1 <= X		R1 <= X		R1 <= X		R1 <= X
	⇓		⇓		⇓		⇓
T + 2	<cond>?R0 <= R1		<cond>?R0 <= R1		<cond>?R0 <= R1		<cond>?R0 <= R1

- At time T+0: The X register reads data form the X register on the PE to the East. This shifts data to the left (or West).
- At time T+1: The R1 register unconditionally reads the data off the shift network (X register)
- At time T+2: The R0 register conditionally loads the data from R1. (i.e. if <cond>=1).

[0069] The timing of the loading is achieved by maintaining a current count in a local counter, which is typically implemented in software. In one embodiment, the local counter is set to an initial value. The local counter can be set in a variety of ways, including loading the counter with the initial value or calculating the initial value locally based on the processing element's location in the matrix (or array) and the function being performed on the data. Thereafter, at certain points in the shifting process, the counter is decremented. For example, the counter may be decremented once for each shift that occurs, or may be decremented once per n clock cycles where n clock cycles equals one shift. As stated, the initial value of the counter depends on its position in the matrix or array and is given by the general function f (Row_Index , Col_Index), where the exact form of f( ) will depend on the particular array manipulation required. When the counter reaches a non-positive value (i.e., zero or negative) the PE selects the data to be loaded into the output matrix.

[0070] Other ways of achieving the same result include resetting the counter to zero and loading each PE with a target value. Thereafter, the counter is incremented producing a current count. When the current count equals the target value, the data value is selected as the final output value to be loaded into the output matrix. Generally, a counter is set to a first known value. Then, at certain programmable points in the algorithm, the value of the counter may be altered, up or down, by a programmable amount. Storing, occurs when a current count in the counter hits a pre-defined target value.

[0071] FIG. 15 illustrates a matrix of data while FIGs. 16A and 16B illustrate two possible transposes of the matrix of data illustrated in FIG. 15. In each case, the leading diagonals (shown in bold) remain unchanged. A matrix of data may be represented by PEs arranged in rows and columns extending in the x and y directions, respectively, or PEs arranged in rows extending in the x and z directions and in columns. The two examples of FIGs. 16A and 16B can be viewed as using different representations for row order. The present invention need not comply to any conventions for it to work; the present invention need only be internally consistent.

[0072] Turning now to FIGs. 17A through 17D, the steps for one embodiment of the method of the present invention of obtaining the transposed data of FIG. 16A are illustrated. This embodiment is useful if there is no wrap mode on the array. This embodiment requires a stack size equal to one less than the number of PEs on a row (or column), and uses all four shift directions. In FIG. 17A, the process begins with a shifting in the north direction of the data under the shaded diagonal. The processing element in row 0, column 3 receives the data values "e", "i" and "m". In a similar manner, the processing element in row 1, column 2 receives the data "j" and "n" while the processing element in row 2, column 1 receives the data "p". This shifting of data may be thought of as a collection step. After the data has been collected and stored within the processing elements along the diagonal, a second plurality of shift operations is performed in which the data stored along the diagonal is output in a second direction, which is perpendicular to the first direction. For example, as seen in FIG. 17B, the data stored by the processing element in the position row 0, column 3, outputs the data in a last in-first out process so that the data "m" after three data shifts, is seen by the processing element in row 0, column 0 while the data "i" is seen by the processing element in row 0, column 1 and the data "e" is seen by the processing element in row 0, column 2. Each of those processing elements stores the data appropriate for its position. Similarly, the processing elements in row 1, columns 0 and 1 store the data "n" and "j", respectively, while the processing element in row 2, column 0 stores the data "p".

[0073] Thereafter, as shown in FIG. 17C, a third shifting operation is performed such that the data "b" is collected and stored by the processing element in row 1, column 2, the data "c" and "g" is collected and stored by the processing element in row 2, column 1 while the processing element in row 3, column 0 receives and stores the data "l", "h", and "d". The aforementioned data is shifted in a direction which is opposite to the direction shown in FIG. 17A. In FIG. 17D, through a series of shifting and storing operations, the data stored along the diagonal in FIG. 17C is driven out and to the left along rows 1, 2 and 3 to arrive at the data pattern illustrated in FIG. 17D. The fourth shifting operation is in a direction opposite to the shifting operation illustrated in FIG. 17B. The processing elements in row 1, column 3, row 2, columns 3 and 2, and row 3, columns 3, 2 and 1 select and store data as a function of their position. All of the shifting and storing operations may be controlled through the use of local counters as discussed above.

[0074] FIGs. 18A through 18D illustrate another embodiment for the present invention which is similar to the embodiment illustrated in FIG. 17A through 17D. In FIGs. 18A through 18D, data from the upper left-hand corner of the matrix is shifted in a first plurality of shifting operations from west to east to enable data to be collected and stored along the shaded diagonal. The data stored in the shaded diagonal is output from the diagonal in a north to south direction to produce the data configuration in FIG. 18B.

[0075] Continuing with FIG. 18C, the original data in the lower right-hand corner of the matrix is shifted in a third plurality of shifting operations from east to west so as to be stored along the shaded diagonal. In a fourth plurality of shifting operations from south to north, the data is output from the shaded diagonal to produce the data pattern illustrated in FIG. 18D.

[0076] Turning now to FIGs. 19A and 19B, another embodiment of the present invention which may be used to obtain the transposed data illustrated in FIG. 16A is illustrated. This embodiment may be a faster embodiment than the embodiments previously described as it uses only two shift directions which are perpendicular. This embodiment requires a wrap mode and a stack size equal to one less than the number of PEs in a row (or column). In the embodiment of FIGs. 19A and 19B, it is assumed that wrap shifting (north to south, or vice versa, and east to west, or vice versa) is available. In FIG. 19A, a single collection step (i.e. a series of three north shifts) is used to store all the data from each column in the PE along the diagonal for that column. In FIG. 19B, a series of three east west shifts enables all the PEs along each row to see all the necessary column data, and by selecting the appropriate data for its position, the transposed data array may be achieved. Selecting of data as final output data may be controlled by local counters as discussed above. The method of FIGs. 19A and 19B may be used depending on the amount of data in the array and the amount of available data storage along the diagonal. Of course, the embodiment of FIGs. 19A and 19B can be implemented used different directions (e.g. south to north shifts and west to east shifts) and can be used to arrive at the transposed data of FIG. 16B.

[0077] Turning now to FIGs. 20A – 20D, another embodiment of the present invention which may be used to obtain the transposed data illustrated in FIG. 16A is illustrated. This embodiment requires a stack size equal to half the number of PEs on a row (or column). This embodiment also requires the wrap mode and it uses all four shift directions. FIG. 20A illustrates in the left portion the input data, i.e. a collection step, while the right portion illustrates the output data, i.e. a distribution step. In FIG. 20A two pieces of data are collected along the diagonal from each column through a south to north shift. In FIG. 20B, right portion, the collected data are driven out from the diagonal in a west to east direction. Another collection step is illustrated in FIG. 20C, left portion, in which the final piece of data from each column are collected along the diagonal through a north to south shift. The collected data are driven out from the diagonal in the right portion of FIG. 20D through an east to west shift.

[0078] As seen in FIGs. 17A through 17D, 18A through 18D, FIGs. 19A and 19B, and FIGs. 20A through 20D the present invention may be implemented through a plurality of shifting and storing operations which are coordinated to enable data to be stored along a diagonal of processing elements from a first direction or first pair of directions and then to be output from that diagonal in a second direction or second pair of directions, perpendicular to the first or first pair of directions, respectively. The amount of data collected may be dependent on the stack size, i.e. number of memory location, and the manner of collection may be dependent on the physical interconnection of the PEs. Both the shifting and the storing operations may be controlled by maintaining local current counts in the appropriate processing elements. Those of ordinary skill in the art will recognize that any method of maintaining a current count, such as counting up or counting down, testing for values other than non-positive values, e.g., a target value, can be carried out while remaining within the scope of the present invention.

[0079] By using the method of the present invention, PEs within a group of PEs can be individually controlled as to the output value which the PE selects for output into the final matrix. Thus, although all of the PEs are responding to the same command, e.g., an east to west wrap shift, each of the PEs is capable of selecting different data at different points during the execution of the instruction thereby enabling various types of data manipulations. Furthermore, by determining which PEs are active, additional flexibility is provided so that subsets of data can be manipulated.

[0080] Although the figures illustrate a two-dimensional (2D) array connected as a mesh the present invention is applicable to other configurations. Accordingly, the phrase "plurality of processing elements" is to be broadly construed to include 2D and 3D collections of PEs connected in any known manner. For example, the PE's could be connected in shapes other than as illustrated in the figures, e.g., a cube. That would have f(x_Index, y_Index, z_Index). An n-dimensional hypercube would have n dimensions and f (d(0), d(1), d(2) .... d(n-1)).

[0081] Additionally, the network need not be connected as a mesh. For example, a simple extension may be implemented by providing two extra connections, one to the PE halfway across the row and the other to the PE halfway down the column. For that example there would be two more shift connections. In addition to the north, east, south, and west shifts, there could also be Half_Row and Half_Col shifts. Both of the above changes could be used at the same time. For example, a four dimensional hyper-cube with half-way connections would have twelve shift options. However, there are only certain configurations of PE's in which a transpose operation makes geometric sense. Consider a 3D cuboid of 8x8x4 PE's. The 3-D transpose would be performed using a 'plane of diagonals'. There are 4 slices of 8x8 PE's, within each slice, the diagonal is used as described above to perform the transpose. Within the cuboid, these diagonals on each of the 4 slices sit on top of each other to form a 4x8 plane of PE's. In general an N-dimensional mesh of PE's that can be 'sliced' to produce a set of MxM squares is amenable to a N-dimensional transpose. Furthermore, certain such meshes may be sliced in different directions to produce different sets of squares, e.g. an 8x8x4x4 mesh could be sliced to produce 8x8 or 4x4 squares.

[0082] While the present invention has been described in connection with a preferred embodiment thereof, those of ordinary skill in the art will recognize that many modifications and variations such as those previously discussed are possible. The present invention is not to be limited by the foregoing description but only by the following claims.